Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 35 (2014) 328 – 337

Procedia Computer Science

18th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems - KES2014

Tightening upper bounds to the expected support for uncertain frequent pattern mining

Carson K. Leung*, Richard Kyle MacKinnon, Syed K. Tanbeer

Department of Computer Science, University of Manitoba, Winnipeg, MB, R3T 2N2, Canada

Abstract

Due to advances in technology, high volumes of valuable data can be collected and transmitted at high velocity in various scientific and engineering applications. Consequently, efficient data mining algorithms are in demand for analyzing these data. For instance, frequent pattern mining discovers implicit, previously unknown, and potentially useful knowledge about relationships among frequently co-occurring items, objects and/or events. While many frequent pattern mining algorithms handle precise data, there are situations in which data are *uncertain*. In recent years, tree-based algorithms for mining uncertain data have been developed. However, tree structures corresponding to these algorithms can be large. Other tree structures for handling uncertain data may achieve compactness at the expense of loose upper bounds on expected supports. In this paper, we propose (i) a compact tree structure for capturing uncertain data, (ii) a technique for using our tree structure to tighten upper bounds to expected support, and (iii) an algorithm for mining frequent patterns based on our tightened bounds. Experimental results show the benefits of our tightened upper bounds to expected supports in uncertain frequent pattern mining.

Peer-review under responsibility of KES International.

Keywords: Data mining; data structure; expected support; frequent pattern mining; knowledge discovery; tree-based mining; uncertain data

1. Introduction

Due to advances in technology, high volumes of valuable data can be collected and transmitted at high velocity in various scientific and engineering applications [10]. Useful knowledge is embedded in these data. Data mining techniques help analyze these data for the discovery of implicit, previously unknown, and potentially useful knowledge (e.g., classifiers [19,24], clusters [20], recommendations [6,21], co-authorship relationships [12], social network patterns [22], frequent patterns). Since the advent of frequent pattern mining [1], numerous studies have been conducted to find *frequent patterns* (i.e., *frequent itemsets*) from *precise* data such as databases of shopper market basket transactions [9]. When mining precise data, users definitely know whether an item is present in (or is absent from) a transaction. In this notion, each item in a transaction t_j in databases of precise data can be viewed as an item with a 100% likelihood of being present in t_j. However, there are situations in which users are uncertain about the presence or absence of

* Corresponding author.
E-mail address: kleung@cs.umanitoba.ca (C.K. Leung)

Peer-review under responsibility of KES International.
doi:10.1016/j.procs.2014.08.113

items[5,7,15]. For example, a meteorologist may suspect (but cannot guarantee) that severe weather phenomena will develop during a thunderstorm. The uncertainty of such suspicions can be expressed in terms of *existential probability*. For instance, a thunderstorm may have a 75% likelihood of generating hail, and only a 20% likelihood of generating a tornado, regardless of whether or not there is hail.

To deal with these situations, a few algorithms have been proposed for mining frequent patterns from (i) dynamic uncertain data streams[13,14] or (ii) static uncertain databases[3,4,17] such as the UF-growth algorithm[16]. In order to compute the *exact* expected support of each pattern, paths in the corresponding UF-tree are shared only if tree nodes on the paths have the same item and the same existential probability values. The resulting UF-tree may be quite large when compared to the FP-tree[8] (for capturing precise data). In an attempt to make the tree compact, the UFP-growth algorithm[2] groups *similar* nodes (with the same item but similar existential probability values) into a cluster. However, depending on the clustering parameter, the corresponding UFP-tree may be as large as the UF-tree. Moreover, because UFP-growth does not store every existential probability value for an item in a cluster, it returns not only the frequent patterns but also some infrequent patterns (i.e., false positives). As alternatives to trees, hyperlinked array structures were used by the UH-Mine algorithm[2], which was reported[23] to outperform UFP-growth. The PUF-growth algorithm[18] was proposed to utilize a concept of an upper bound to expected support together with more aggressive path sharing to yield a more compact tree structure, and it was shown to outperform UH-Mine.

Here, we examine (i) how to further tighten the upper bound on expected support? We also examine (ii) how to make the resulting tree as compact as the FP-tree? and (iii) how to mine frequent patterns from such a tree? Our key contributions of this paper are as follows:

1. the concept of the tightened prefixed pattern cap (TPC);
2. a tightened prefixed-capped uncertain frequent pattern tree (TPC-tree) structure, which can be as compact as the original FP-tree while capturing uncertain data; and
3. a tightened prefixed-capped uncertain frequent pattern-growth mining algorithm—called TPC-growth—which is guaranteed to mine *all and only those* frequent patterns (i.e., *no* false negatives and *no* false positives) from uncertain data.

The remainder of this paper is organized as follows. The next section gives background and related works. Section 3 discusses how the TPC tightens the upper bounds to the expected support. In Sections 4 and 5, we present our TPC-tree structure and TPC-growth algorithm, respectively. Evaluation results are shown in Section 6, and conclusions are given in Section 7.

2. Background and related works

We first give some background information about frequent pattern mining of uncertain data (e.g., existential probability, expected support), and we then discuss some related works.

2.1. Existential probability and expected support

Here, we provide some background information about (i) the existential probability (which expresses the uncertainty of suspicions) and (ii) the expected support (which can be computed based on existential probabilities).

Definition 1. Let (i) `Item` be a set of m domain items and (ii) $X = \{x_1, x_2, \ldots, x_k\}$ be a pattern comprising k items (i.e., a *k-itemset*), where $X \subseteq$ `Item` and $1 \leq k \leq m$. Then, each item x_i in a transaction $t_j = \{x_1, x_2, \ldots, x_h\} \subseteq$ `Item` in a transactional database of uncertain data is associated with an *existential probability* $P(x_i, t_j)$[11] with value

$$0 < P(x_i, t_j) \leq 1, \tag{1}$$

where $P(x_i, t_j)$ represents the likelihood of the presence of x_i in t_j. □

With the above definition, the *existential probability* $P(X, t_j)$ of a pattern X in t_j is then the product of the corresponding existential probability values of every item x within X (where these items are independent)[11]:

$$P(X, t_j) = \prod_{x \in X} P(x, t_j), \tag{2}$$

Table 1. A transactional database of uncertain data (*minsup*=1.1).

TID	Contents of each transaction
t_1	*a*:0.9, *b*:0.3, *c*:0.1, *d*:0.9, *e*:0.6
t_2	*a*:0.5, *b*:0.2, *c*:0.1, *d*:0.9, *f*:0.6
t_3	*a*:0.6, *b*:0.1, *c*:0.2, *e*:0.8, *f*:0.5
t_4	*a*:0.7, *b*:0.2, *c*:0.2, *e*:0.9
t_5	*b*:0.9, *c*:0.9, *g*:0.4

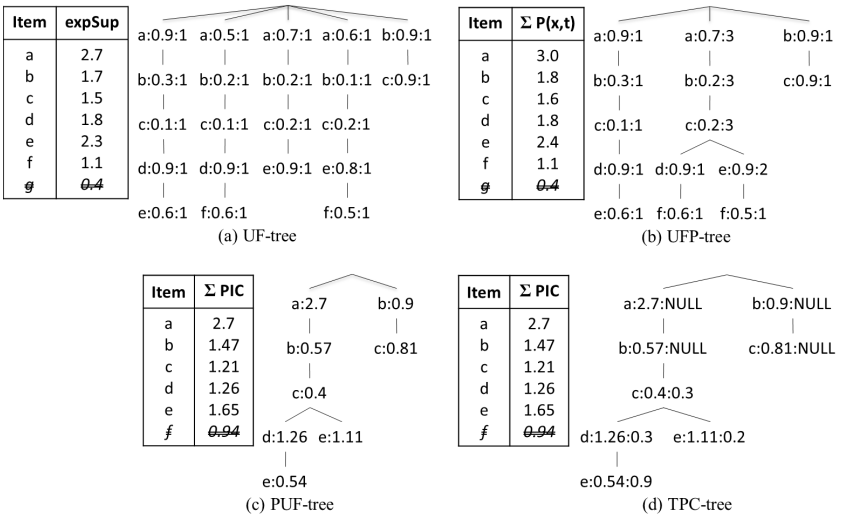


Fig. 1. The UF-tree[16], UFP-tree[2], PUF-tree[18], and our proposed TPC-tree for uncertain data in Table 1 when *minsup*=1.1.

where $P(x, t_j)$ is the existential probability value of x in t_j.

Definition 2. The *expected support expSup*(X) of a pattern X in the database of uncertain data is the sum of existential probability $P(X, t_j)$ of X in transaction t_j over all n transactions in the database:

$$expSup(X) = \sum_{j=1}^{n} P(X, t_j) = \sum_{j=1}^{n} \left(\prod_{x \in X} P(x, t_j) \right), \tag{3}$$

when items $x \in X$ in every transaction t_j are independent. □

Given (i) a database of uncertain data and (ii) a user-specified minimum support threshold *minsup*, the research problem of *frequent pattern mining from uncertain data* is to discover from the database all those *frequent* patterns (i.e., patterns having expected support ≥ *minsup*).

2.2. Existing tree-based frequent pattern mining algorithms: UF-growth, UFP-growth and PUF-growth

To mine frequent patterns from uncertain data, the *UF-growth algorithm*[16] scans the data twice to build a *UF-tree*. Each node in a UF-tree captures (i) an item x, (ii) its existential probability, and (iii) its occurrence count. Tree paths are shared if the nodes on these paths share the same item and existential probability. In general, when dealing with uncertain data, it is not uncommon that the existential probability values of the same item vary from one transaction to another. As such, the resulting UF-tree may not be as compact as the FP-tree. Fig. 1(a) shows a UF-tree for the uncertain data presented in Table 1 when *minsup*=1.1. The UF-tree contains four nodes for item *a* with different probability values as children of the root. Efficiency of the corresponding UF-growth algorithm, which finds all and *only those* frequent patterns, partially relies on the compactness of the UF-tree.

To attempt making the tree more compact, the *UFP-growth algorithm*[2] builds a *UFP-tree* (by also scanning the uncertain data twice). Tree paths are shared if the nodes on these paths share the same item but *similar* existential probability values. With such a less restrictive path sharing condition, nodes for item x having similar existential probability values are clustered into a mega-node. The resulting mega-node in the UFP-tree captures (i) an item x, (ii) the *maximum* existential probability value (among all nodes within the cluster), and (iii) its occurrence count. See Fig. 1(b). By extracting appropriate tree paths and constructing UFP-trees for subsequent projected databases, the UFP-growth algorithm finds all frequent patterns and some *false positives* at the end of the second scan of uncertain data. A third scan is then required to remove those false positives.

To further attempt in improving the compactness of the tree, the *PUF-growth algorithm*[18] uses a *PUF-tree* to tighten the upper bound on the expected support of patterns. Each node in a PUF-tree captures (i) an item x and (ii) a *prefixed item cap* (*PIC*). See Fig. 1(c) for a PUF-tree, which represents the same database of uncertain data as the UF-tree in Fig. 1(a).

Definition 3. The *prefixed item cap* (*PIC*)[18] of an item x_r in a transaction $t_j = \{x_1, \ldots, x_r, \ldots, x_h\}$ where $1 \leq r \leq h$—denoted as $PIC(x_r, t_j)$—is defined as the product of (i) $P(x_r, t_j)$ and (ii) the highest existential probability value M_1 of items from x_1 to x_{r-1} in t_j—i.e., in the *proper "prefix"* of x_r in (the "ordered"[†]) t_j:

$$PIC(x_r, t_j) = \begin{cases} P(x_1, t_j) & \text{if } h = 1 \\ P(x_r, t_j) \times M_1 & \text{if } h > 1 \end{cases} \quad (4)$$

where $M_1 = \max_{1 \leq q \leq r-1} P(x_q, t_j)$. □

The PUF-growth algorithm mines frequent patterns by taking advantage of the tree structure to restrict the computation of upper bounds of expected support to the highest existential probability among items in the *"prefix"* of x via the use of the PIC. See Example 1. Direct benefits include fewer false positives and shorter mining time because fewer projected databases are needed to be extracted and less work is required in a third scan of the uncertain data.

Example 1. Consider an "ordered" transaction $t_1 = \{a{:}0.9, b{:}0.3, c{:}0.1, d{:}0.9, e{:}0.6\}$ in Table 1. If $X{=}\{a, b, c, d\}$, then $PIC(d, t_1) = P(d, t_1) \times M_1 = 0.9 \times 0.9 = 0.81$.

This PIC also serves as an upper bound to the expected support of $X = \{a, d\}, \{b, d\}, \{c, d\}, \{a, b, d\}, \{a, c, d\}, \{b, c, d\}$, or $\{a, b, c, d\}$. While this upper bound is tight for short patterns like $\{a, d\}$ having $P(\{a, d\}, t_1){=}0.81$, it becomes loose for long patterns like $\{a, b, d\}$ having $P(\{a, b, d\}, t_1){=}0.243$ and $\{a, b, c, d\}$ having $P(\{a, b, c, d\}, t_1){=}0.0243$. □

As observed from the above example, an upper bound based on PIC may not be too tight when dealing with long patterns mined from long transactions of uncertain data. In many real-life situations, it is not unusual to have long patterns to be mined from long transactions of uncertain data.

3. Our tightened prefixed pattern cap (TPC) for tightening upper bounds to the expected support

To tighten the upper bound for patterns of all cardinality k (i.e., k-itemsets for $k \geq 2$), we propose the concept of a *tightened prefixed pattern cap* (*TPC*). The key idea is to keep track of a new value—a *"silver" value*—which is the second highest probability value M_2 in the "prefix" of t_j. Every time a frequent extension ($k > 2$) is added to the suffix item x_r, this "silver" value is used. As a preview, each node in the corresponding tree structure contains (i) an item x_r, (ii) its PIC, and (iii) its "silver" value. See the following definitions, examples, and observations.

Definition 4. Let (i) $t_j = \{x_1, \ldots, x_r, \ldots, x_h\}$ where $1 \leq r \leq h$, (ii) $X = \{y_1, y_2, \ldots, y_k\}$ is a k-itemset in t_j such that $y_k = x_r$, and (iii) M_2 denoting the "silver" value be the second highest existential probability value of items from x_1 to x_{r-1} in t_j (i.e., in the *proper "prefix"* of x_r in t_j). Then, the *tightened prefixed pattern cap* (*TPC*) is defined as follows:

$$TPC(X, t_j) = \begin{cases} PIC(x_r, t_j) & \text{if } k \leq 2 \\ PIC(x_r, t_j) \times \prod_{i=3}^{k} M_2 \;=\; PIC(x_r, t_j) \times M_2^{k-2} & \text{if } k \geq 3 \end{cases} \quad (5)$$

where $PIC(x_r, t_j)$ is the prefixed item cap of x_r in t_j as defined in Definition 3. □

[†] When applying a tree-based frequent pattern mining algorithm (e.g., PUF-growth), items in transactions are usually arranged in some predefined order along tree paths.

Example 2. Revisit Example 1 by reconsidering the "ordered" transaction t_1 = {a:0.9, b:0.3, c:0.1, d:0.9, e:0.6} in Table 1. If X={a, b, c, d}, then $TPC(X, t_1) = PIC(d, t_1) \times M_2^2 = 0.81 \times 0.3^2 = 0.0729$, which is much closer to its $P(X, t_1)$ =0.0243 when compared with the old bound of 0.81 provided by $PIC(d, t_1)$.

Similarly, if X={a, b, d}, then $TPC(X, t_1) = PIC(d, t_1) \times M_2 = 0.81 \times 0.3 = 0.243$, which is as tight as its $P(\{a, b, d\}, t_1)$ = 0.243.

Moreover, if X={a, d}, then $TPC(X, t_1) = PIC(d, t_1) = 0.81$, which again is as tight as its $P(\{a, d\}, t_1)$=0.81. □

Definition 5. The *cap of expected support expSupCap*(X) of a pattern $X = \{y_1, \ldots, y_k\}$ (where $k > 1$) is defined as the sum (over all n transactions in a database) of all the TPCs of X:

$$expSupCap(X) = \sum_{j=1}^{n} \left\{ TPC(X, t_j) \mid X \subseteq t_j \right\}, \tag{6}$$

where $TPC(X, t_j)$ is the TPC of X in a transaction t_j as defined in Definition 4. □

Observation 1. Based on Definition 5, *expSupCap*(X) serves as an *upper bound* to the expected support of X, i.e., $expSup(X) \leq expSupCap(X)$. We observed the following:

(a) If $expSupCap(X) < minsup$, then X cannot be frequent. Conversely, if X is a frequent pattern, then $expSupCap(X)$ must be $\geq minsup$. Such a *safe/sound condition*—with respect to $expSupCap(X)$ and $minsup$—can be safely applied to mining all frequent patterns for data analytics.

(b) The expected support $expSup(X)$ satisfies the *downward closure property*[1] as $expSup(X) \leq expSup(Y)$ for all $Y \subset X$. So, (i) $expSup(X) \geq minsup$ implies $expSup(Y) \geq minsup$, and (ii) $expSup(Y) < minsup$ implies $expSup(X) < minsup$.

(c) The cap of expected support $expSupCap(X)$ of any pattern X based on the TPC does *not* always satisfy the downward closure property. As an example, $expSupCap(\{a, b, c\})$=0.077 < 0.0909=$expSupCap(\{a, b, c, d\})$ in Table 1.

(d) For special cases where X and its subset Y sharing the same suffix item (e.g., Y={a, b, d} ⊂ {a, b, c, d}=X sharing the suffix item d), the *cap of expected support* based on the TPC satisfies the downward closure property. We call this property the *partial* downward closure property. □

4. Our TPC-tree structure for capturing important contents of uncertain data

As the TPC provides a tighter upper bound to the expected support, we propose a *tightened prefixed-capped uncertain frequent pattern tree* (*TPC-tree*) *structure* to efficiently capture contents of uncertain data so that the TPC can be computed based on Eq. (5) using the PIC and the "silver" value M_2. Specifically, each node in this TPC-tree structure contains (i) an item x_r, (ii) its PIC, and (iii) its M_2. See Fig. 1(d).

To construct a TPC-tree, we first scan the database of uncertain data. By doing so, we find all distinct frequent items and construct a header table called an *item-list* to store only frequent items in some consistent order (e.g., canonical order) to facilitate tree construction. Then, the TPC-tree is constructed with the second database scan in a fashion similar to that of the FP-tree[8]. A key difference is that, when inserting a transaction item, we compute both its PIC and M_2 values. The item is then inserted into the TPC-tree according to the ordering in the item-list. If a node containing that item already exists in the tree path, we update (i) its PIC by *summing* the computed PIC value with the existing one and (ii) its M_2 value by taking the *maximum* between the computed M_2 value and the existing one. Otherwise, we create a new node with the computed PIC and M_2 values. For a better understanding of the TPC-tree construction, see Example 3.

Example 3. Consider the database of uncertain data in Table 1. Let (i) the user-specified support threshold *minsup* be set to 1.1; let (ii) the item-list follow the alphabetical ordering of items. After the first database scan, the contents of the item-list after computing the expected supports of all items and after removing infrequent items (e.g., item g) are ⟨a:2.7, b:1.7, c:1.5, d:1.8, e:2.3, f:1.1⟩.

With the second database scan, we insert only the frequent items of each transaction (with their respective PIC and M_2 values) in the ordering of the item-list. For instance, when inserting transaction t_1={a:0.9, b:0.3, c:0.1, d:0.9,

e:0.6}, items a, b, c, d and e—with their respective PIC and (if appropriate) M_2 values such as $\langle 0.9, \text{NULL}\rangle$ for a, $\langle 0.3\times0.9=0.27, \text{NULL}\rangle$ for b, $\langle 0.1\times0.9=0.09, 0.3\rangle$ for c, $\langle 0.9\times0.9=0.81, 0.3\rangle$ for d, $\langle 0.6\times0.9=0.54, 0.9\rangle$ for e) are inserted in the TPC-tree. As t_2 shares a common "prefix" $\langle a, b, c, d\rangle$ with an existing path in the TPC-tree created when t_1 was inserted, (i) the PIC values of those items in the common "prefix" (i.e., a, b, c and d) are added to their corresponding nodes (e.g., 0.9+0.5=1.4 for a, 0.27+0.1=0.37 for b, 0.09+0.05=0.14 for c, 0.81+0.45=1.26 for d), (ii) the M_2 values of those items are checked against the existing M_2 values for their corresponding nodes, with only the maximum saved for each node (e.g., max{0.3, 0.2}=0.3 for c, max{0.3, 0.2}=0.3 for d), and (iii) the remainder of the transaction (i.e., a new branch for item f) is inserted as a child of the last node of the "prefix" (i.e., as a child of d). Fig. 1(d) shows the TPC-tree after inserting all the transactions and pruning those items with infrequent extensions (e.g., item f because its *expSupCap*($\{f\}$)—provided by the total TPC value—is less than the user-specified *minsup*. See Observation 2. Similar to other tree structures for frequent pattern mining (e.g., FP-tree), our TPC-tree maintains horizontal node traversal pointers, which are not explicitly shown in the figures for simplicity. □

Observation 2. Based on the aforementioned process of constructing a TPC-tree, we observed the folllowing:

(a) Although we arranged all items in canonical order when inserting them into the TPC-tree in Example 3, we could also use other orderings (e.g., descending order of expected support or occurrence counts). If we were to store items in descending order of occurrence counts, then *the number of nodes in the resulting TPC-tree would be the same as that of the FP-tree*[8].

(b) We can similarly remove any item having the sum of PIC values (in the item-list) less than *minsup* because it is guaranteed to have no frequent extensions. Hence, we can remove item g from the TPC-tree in Example 3 because the sum of PIC values of g is less than *minsup*. This *tree-pruning technique* saves mining time as it skips all k-itemsets (for $k \geq 2$) with suffix g as they are all infrequent.

(c) The PIC value in a node x in a TPC-tree maintains the sum of PIC values of an item x for all transactions that pass through or end at x. Because common "prefixes" are shared, the *TPC-tree becomes more compact than the UFP-tree*[2] and avoids having siblings (nodes with the same parent node) containing the same item but having different existential probability values.

(d) As the TPC-tree captures all frequent items in every transaction of uncertain data and stores their PIC & M_2 values, frequent pattern mining based on the TPC computed using PIC & M_2 values ensures that no frequent patterns will be missed (i.e., *no false negatives*).

(e) Based on Eq. (3), the expected support of $X = \{x_1, \ldots, x_k\}$ is computed by summing $P(X, t_j)$ of every t_j, where $P(X, t_j)$ is the product of the existential probability value of x_k with those of other items in the proper "prefix" of X, i.e., $P(X, t_j) = P(x_k, t_j) \times \left(\prod_{i=1}^{k-1} P(x_i, t_j)\right)$. Based on Eq. (4), the PIC is computed based on the existential probability value of x_k and the *single highest* existential probability value M_1 in its "prefix": $PIC(x_k, t_j) = P(x_k, t_j) \times M_1$. In contrast, based on Eq. (5), the TPC for X—computed based on the existential probability value of x_k and the *two highest* existential probability values M_1 & M_2 in its "prefix"—provides a *tighter upper bound* because the TPC tightens the bound as potentially frequent patterns are generated during the mining process with increasing cardinality of X, whereas the PIC has no such compounding effect:

$$P(X, t_j) \;\leq\; TPC(X, t_j) \;\leq\; PIC(x_k, t_j) \tag{7}$$

as $\left(P(x_k, t_j)\times\prod_{i=1}^{k-1} P(x_i, t_j)\right) \leq \left(P(x_k, t_j)\times M_1\times M_2^{k-2}\right) \leq \left(P(x_k, t_j)\times M_1\right)$. □

5. Our TPC-growth algorithm for mining frequent patterns from uncertain databases

Next, we propose a *tightened prefixed-capped uncertain frequent pattern-growth mining algorithm* (*TPC-growth*), which finds frequent patterns from our TPC-tree structure that captures uncertain data. Recall from Section 4 that the construction of a TPC-tree is similar to that of a PUF-tree, except that "silver" values are additionally stored. Thus, the basic operation in TPC-growth is to construct a projected database for each potential frequent pattern and recursively mine its potentially frequent extensions.

If an item x is found to be potentially frequent, its existential probability must contribute to the expected support computation for every pattern constructed from its $\{x\}$-projected database (denoted as DB_x). Hence, *expSupCap*($\{x\}$)

based on the TPC is guaranteed to be the upper bound of the expected support of any pattern with suffix x due to Eq. (7). Theoretically, this implies that the complete set of patterns with suffix x can be mined based on the partial downward closure property (Observation 1(d)). Practically, we can directly proceed to generate all potentially frequent patterns from the TPC-tree because $expSupCap(Y \cup X)$ in the original database $\geq minsup$ if and only if $expSupCap(Y)$ in the X-projected database $DB_X \geq minsup$ (where $Y \in DB_X$ and $expSupCap(X) \geq minsup$).

By doing so, we find every pattern X with $expSupCap(X) \geq minsup$. Like UFP-growth[2] and PUF-growth[18], our TPC-growth mining process may generate some false positives at the end of the second database scan, and all these false positives will be filtered out with the third database scan. Hence, our TPC-growth algorithm is guaranteed to return *all* and *only those* frequent patterns with *neither* false positives *nor* false negatives.

Example 4. The TPC-growth algorithm mines extensions of every item in the item-list/header. With when *minsup*=1.1, the $\{e\}$-conditional tree is constructed by extracting the tree paths $\langle a$:2.7:NULL, b:0.57:NULL, c:0.4:0.3, d:1.26:0.3, e:0.54:0.9$\rangle$ and $\langle a$:2.7:NULL, b:0.57:NULL, c:0.4:0.3, e:1.11:0.2$\rangle$. When projecting these two paths, TPC-growth computes the cap of expected support for each item in the projected database using the PIC and M_2 values from all f nodes in the original tree.

This $\{e\}$-conditional tree is then used to generate (i) all 2-itemsets containing item e and (ii) their further extensions by recursively constructing projected databases from them. For all k-itemsets (where $k \geq 3$) that are generated, the cap of expected support is multiplied by the M_2 value. Consequently, for cardinality $k = 2$, potentially frequent patterns $\{a, e\}, \{b, e\}, \{c, e\}$ & $\{d, e\}$ are generated because all of them have their caps of expected support equal to $0.54 + 1.11 = 1.65$. However, unlike PUF-growth, no potentially frequent patterns of higher cardinality are generated with this suffix. For instance, we do not generate $\{a, b, e\}, \{a, c, e\}$ & $\{b, c, e\}$ because their caps of expected support equal to $(0.54 \times 0.9) + (1.11 \times 0.2) < minsup$. We also do not generate $\{a, d, e\}, \{b, d, e\}$ & $\{c, d, e\}$ because their caps are even lower.

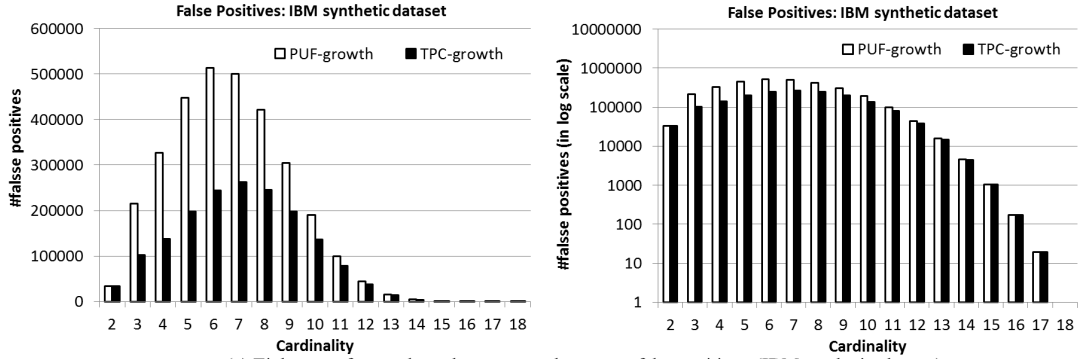
Patterns ending with items b, c and d can then be mined in a similar fashion. The complete set of potentially frequent patterns generated by TPC-growth includes $\{b, c\}$:1.21, $\{a, d\}$:1.26, $\{b, d\}$:1.26, $\{c, d\}$:1.26, $\{a, e\}$:1.65, $\{b, e\}$:1.65 & $\{c, e\}$:1.65. All of them are then checked against the database to find those truly frequent ones (after a third scan). □

As shown in Example 4, TPC-growth finds a complete set of patterns from a TPC-tree without any false negatives. In addition, with the small concession of storing one extra value in each node (i.e., the "silver" value), TPC-growth does so while generating fewer false positives than PUF-growth. In much larger databases this effect has a huge impact on the number of the false positives generated and thus directly results in lower runtimes.

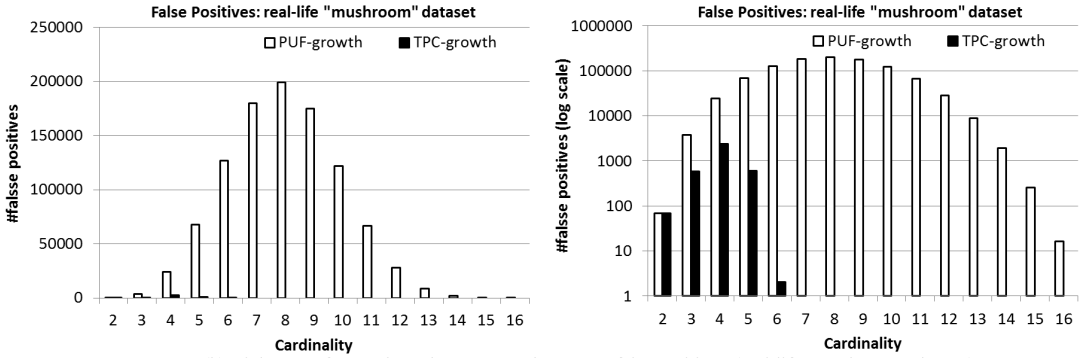
6. Evaluation results

For evaluation, we compared the performances of our TPC-growth algorithm with the existing PUF-growth[18] algorithm, which was shown to outperform UF-growth[16], UFP-growth[2] and UH-Mine[2]. We used both synthetic and real-life datasets for our tests. The synthetic datasets, which are generally sparse, were generated within a domain of 1000 items by the data generator developed at IBM Almaden Research Center[1]. We also considered several real-life datasets such as kosarak, mushroom and retail. We assigned a (randomly generated) existential probability value from the range (0,1] to each item in every transaction in these datasets. The name of each dataset indicates some characteristics of the dataset. For example, the dataset u100K_5L_10_100 contains 100K transactions with average transaction length of 5, and each item in a transaction is associated with an existential probability value that lies within a range of [10%, 100%].

All programs were written in C++ and ran in a Linux environment on an Intel Core i5-661 CPU with 3.33 GHz and 7.5 GB RAM. Unless otherwise specified, runtime includes CPU and I/Os for item-list construction, TPC-tree construction, mining, and false-positive removal. While the number of false positives generated at the end of the second database scan may vary, all algorithms (ours and others) produce the same set of truly frequent patters at the end of the mining process. The results shown in this section are based on the average of multiple runs for each case. In all experiments, *minsup* was expressed in terms of the absolute support value, and all trees were constructed using the ascending order of item value.



(a) Tightness of upper bounds to expected support: false positives (IBM synthetic dataset)



(b) Tightness of upper bounds to expected support: false positives (real-life "mushroom" dataset)

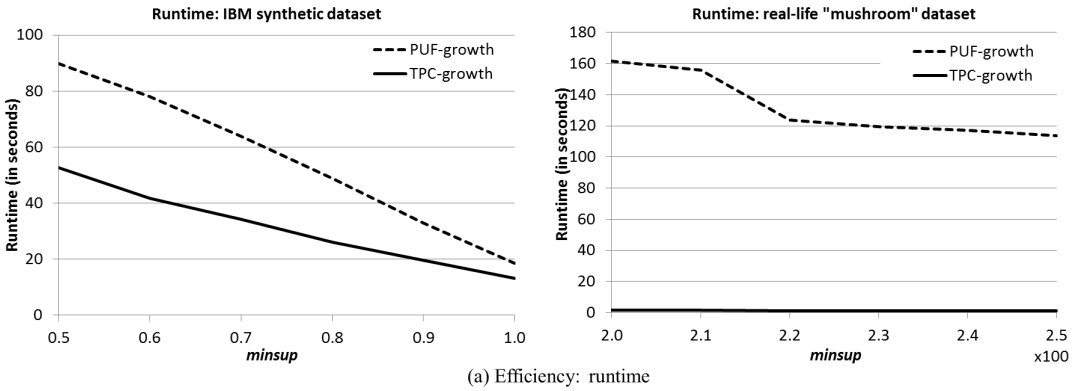
Fig. 2. Experimental results: tightness of upper bounds to expected support (number of false positives).

6.1. Tightness of upper bounds to expected support: reduction in the number of false positives

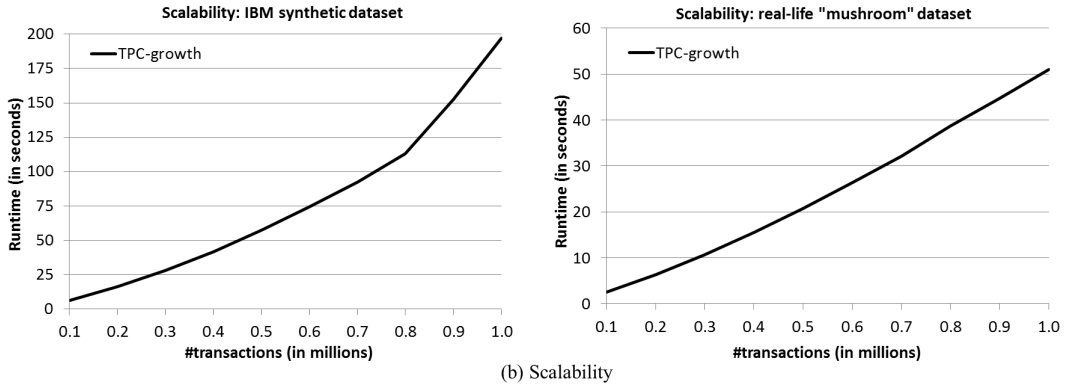
To measure the tightness of upper bounds to expected support, we compared the number of false positives generated by the existing PUF-growth algorithm[18] with that of our TPC-growth algorithm. Their overall performances depend on the number of false positives generated. In this experiment, we measured the number of false positives generated by both algorithms for fixed values of *minsup* with different datasets. Here, we present results using one *minsup* value for each of the two datasets (i.e., u100K_5L_10_100) and mushroom_50_60 in Figs. 2(a)–(b). Note that, although TPC-trees take up more space (as they capture three components per node) than PUF-trees (as they capture two components per node), TPC-growth was observed to significantly reduce the number of false positives when compared with PUF-growth. The primary reason for this improvement is that the upper bounds for the TPC-growth algorithm are much tighter than PUF-growth for patterns of higher cardinality k (where $k > 2$), and thus fewer potentially frequent patterns are generated and subsequently fewer false positives. As shown in Fig. 2(a), TPC-growth generated around 50% of the false positives generated by PUF-growth. Moreover, when existential probability values were distributed over a narrow range with a higher *minsup* as shown in Fig. 2(b), TPC-growth generated (i) only 1.6% of the false positives generated by PUF-growth when $3 \leq k \leq 6$ and (ii) *no* false positives when $k \geq 7$. In total, TPC-growth generated only 0.36% of false positives generated by PUF-growth. Furthermore, TPC-growth required shorter runtimes than PUF-growth in every single experiment we ran.

6.2. Efficiency and scalability of the corresponding uncertain frequent pattern mining algorithm

As PUF-growth was shown[23] to outperform UH-Mine[18] and UFP-growth[2], we compared our TPC-growth algorithm with PUF-growth. Fig. 3(a) shows that TPC-growth required shorter runtimes than PUF-growth for datasets mushroom_50_60 and u100K_5L_10_100. The primary reason is that, even though PUF-growth finds all frequent patterns when mining an extension of X, it may suffer from the high computation cost of generating unnecessarily large



(a) Efficiency: runtime



(b) Scalability

Fig. 3. Experimental results: efficiency (runtime) and scalability.

numbers of potentially frequent patterns as it only uses $P(x_r, t_j)$ and the single highest existential probability value M_1 in the "prefix" of x_r in t_j in its PIC calculation. This allows large numbers of potentially frequent patterns of high cardinality to be generated with similar expected support cap values to those of low cardinality having the same suffix item. The use of the self-product of M_2 in TPC-growth ensures that those patterns with high cardinality are never generated due to their expected support caps being much closer to the expected support. This effect becomes more pronounced with lower *minsup* values, widening the gap in runtimes even further between the two algorithms. Since the TPC calculation in TPC-growth becomes closer to the true expected support value as the cardinality of potentially frequent patterns under consideration is increased, lower *minsup* values have a much smaller effect on increasing run-times in TPC-growth than in PUF-growth.

Given that high volumes of high-variety, high-veracity and valuable data can be collected and transmitted at high velocity, we also evaluated the scalability of TPC-growth. We applied the algorithm to mine frequent patterns from datasets with increasing size. The experimental results presented in Fig. 3(b) demonstrate that our algorithm (i) is scalable with respect to the number of transactions and (ii) can mine high volumes of uncertain data within a reasonable amount of time. The experimental results show that our TPC-growth algorithm effectively mines frequent patterns from uncertain data irrespective of distribution of existential probability values (whether most of them have low or high values and whether they are distributed into a narrow or wide range of values).

7. Conclusions

In this paper, we proposed the concept of TPC, which tightens the upper bound to the expected support of frequent patterns to be mined from uncertain data. The TPC is computed based on the information captured by the TPC-tree structure. Once such a TPC-tree structure is constructed by the TPC-growth algorithm (after two scans of the uncertain

data), all potentially frequent patterns—containing *all* truly frequent patterns (i.e., *no* false negatives) but some false positives (i.e., any pattern X with $expSupCap(X) \geq minsup$ but with $expSup(X) < minsup$)—can then be mined from the TPC-tree structure. Fortunately, the number of false positives is reduced as the TPC helps tighten the upper bounds to expected supports. To complete the mining process, TPC-growth scans the uncertain data a third time to compute the true expected support and to eliminate this small number of false positives. Evaluation results show, although TPC-tree takes up more (e.g. 50%) space than the existing PUF-growth algorithm, it pays off because the tightness of these upper bounds produced by the TPC led to a significantly low number (e.g., 1%) of false positives.

Acknowledgements

This project is partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the University of Manitoba.

References

1. Agrawal R, Srikant R. Fast algorithms for mining association rules in large databases. In: *Proceedings of the VLDB 1994*. Morgan Kaufmann; 1994, p. 487–499.
2. Aggarwal CC, Li Y, Wang J, Wang J. Frequent pattern mining with uncertain data. In: *Proceedings of the ACM KDD 2009*. ACM; 2009, p. 29–37.
3. Bernecker T, Kriegel HP, Renz M, Verhein F, Zuefle A. Probabilistic frequent itemset mining in uncertain databases. In: *Proceedings of the ACM KDD 2009*. ACM; 2009, p. 119–127.
4. Budhia BP, Cuzzocrea A, Leung CK. Vertical frequent pattern mining from uncertain data. In: *Proceedings of the KES 2012*. IOS Press; 2012, p. 1273–1282.
5. Calders T, Garboni C, Goethals B. Efficient pattern mining of uncertain data with sampling. In: *Proceedings of the PAKDD 2010, Part I*. Springer; 2010, p. 480–487.
6. Chowdhury NK, Leung CK. Improved travel time prediction algorithms for intelligent transportation systems. In: *Proceedings of the KES 2011, Part II*. Springer; 2011, p. 355–365.
7. Cuzzocrea A, Leung CK, MacKinnon RK. Mining constrained frequent itemsets from distributed uncertain data. *Future Generation Computer Systems* 2014; **37**:117–126.
8. Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation. In: *Proceedings of the ACM SIGMOD 2000*. ACM; 2000, p. 1–12.
9. Lakshmanan LVS, Leung CK, Ng RT. Efficient dynamic mining of constrained frequent sets. *ACM Transactions on Database Systems* 2003; **28**(4):337–389.
10. Leung CK. Big data mining and analytics. In: Wang J, editor. *Encyclopedia of business analytics and optimization*. IGI Global; 2014, p. 328–337.
11. Leung CK. Mining uncertain data. *WIREs Data Mining and Knowledge Discovery* 2011; **1**(4):316–329.
12. Leung CK, Carmichael CL, Teh EW. Visual analytics of social networks: mining and visualizing co-authorship networks. In: *Proceedings of the FAC 2011, HCII 2011*. Springer; 2011, p. 335–345.
13. Leung CK, Hao B. Mining of frequent itemsets from streams of uncertain data. In: *Proceedings of the IEEE ICDE 2009*. IEEE; 2009, p. 1663–1670.
14. Leung CK, Jiang F. Frequent itemset mining of uncertain data streams using the damped window model. In: *Proceedings of the ACM SAC 2011*. ACM; 2011, p. 950–955.
15. Leung CK, Jiang F. Frequent pattern mining from time-fading streams of uncertain data. In: *Proceedings of the DaWaK 2011*. Springer; 2011, p. 252–264.
16. Leung CK, Mateo MAF, Brajczuk DA. A tree-based approach for frequent pattern mining from uncertain data. In: *Proceedings of the PAKDD 2008*. Springer; 2008, p. 653–661.
17. Leung CK, Tanbeer SK. Fast tree-based mining of frequent itemsets from uncertain data. In: *Proceedings of the DASFAA 2012, Part I*. Springer; 2012, p. 272–287.
18. Leung CK, Tanbeer SK. PUF-tree: a compact tree structure for frequent pattern mining of uncertain data. In: *Proceedings of the PAKDD 2013, Part I*. Springer; 2013, p. 13–25.
19. Nakashima T, Sumitani T, Bargiela A. A construction method of fuzzy classifiers using confidence-weighted learning. In: *Proceedings of the KES 2013*. Elsevier; 2013, p. 460–466.
20. Oh CH, Honda K. Dual exclusive partition in fuzzy CoDoK and SCAD-based fuzzy co-clustering. In: *Proceedings of the KES 2013*. Elsevier; 2013, p. 800–809.
21. Su JH, Chang WY, Tseng VS. Personalized music recommendation by mining social media tags. In: *Proceedings of the KES 2013*. Elsevier; 2013, p. 303–312.
22. Tanbeer SK, Leung CK, Cameron JJ. Interactive mining of strong friends from social networks and its applications in e-commerce. *Journal of Organizational Computing and Electronic Commerce* 2014; **24**(2–3):157–173.
23. Tong Y, Chen L, Cheng Y, Yu PS. Mining frequent itemsets over uncertain databases. *PVLDB* 2012; **5**(11):1650–1661.
24. Yang L, Li C, Ding Q, Li L. Combining lexical and semantic features for short text classification. In: *Proceedings of the KES 2013*. Elsevier; 2013, p. 78–86.